Profiling and optimization of `LArSoft` code

Gianluca Petrillo

MicroBooNE Offline Software Retreat
Yale University, March 17th , 2014

What is "profiling"

Profiling:

the assessment of the resources that a program uses

- how much memory the program needs?
- how much disk space?
- how long it takes to run?

The optimization of a program starts from the analysis of the portions of it which take the largest resources:

1. find which part of code (ab)uses resources
2. understand *how* the resources are being used (and why)
3. act: fix the bug, try a different approach or redesign in full

CPU usage profiling

Goal:

Identify the portions of the code which take the longest to run

Using a proper tool, we can discover that `TH1::Fit` takes a lot of time. Thank you.
We want to know who asked for it!

- some tools collect a list of all the callers of the function; now we know that `TH1::Fit` was called by a function using 4% of the total CPU: `hit::GausHitFinder::produce`
- some tools collect the full list of the functions, from `main` down to `TH1::Fit`, which have been called to get there; this is called the call stack
- turns out that `hit::GausHitFinder::produce` calls `TH1::Fit` twice; some tools can tell us how much each single call spent (but sometimes they just guess...): line 472, taking 0.8% of the CPU time, at line 555 with 3%

CPU profiling: options

We have different possibilities to collect that information:

inclusive timing	based on the system clock, it can be the overall run time (e.g. `time` builtin in `bash`) or time report at checkpoints
sampling	peek into the program at random times to see what it is doing (same principle as Monte Carlo integration)
stepping	monitor every function call and how long it takes to complete

CPU profiling tools

`time`	(`bash` command): per-run information
`Timing`	(`art` service): per-event, module-level information
`gperftools`	(Google) quick snapshot of where time is spent with full call history
`callgrind`	(`valgrind` tool) count of each call and used cycles

A number of other tools are available: GNU `gprof` (told not to deal too well with C++11 yet), FermiLab FAST (same features as `gperftools`, but falling behind as the PC architecture evolves), Open|SpeedShop (also same features as `gperftools`, plus a nice interface, but a nightmare to compile), `IgProf` (I haven't tried), ...

art Timing service

What Timing service does

At the end of each art *module*, the time elapsed by it is printed out.
The time is based on the “wall clock”.
It also provides the total time for the event, and a summary in the end.

+ out of the box with art
+ usually FCL files already have it enabled
+ if not, just add `services.Timing: {}` to the configuration

– rough code identification: just tells the module

art Timing service in action

The following timing is part of a 10-event run with 3D reconstruction, (special configuration by Eric Church):

```
lar -c ./standard_reco_uboone_3D_cosmic_eric20140313.fcl \
-s prodgenie_bnb_nu_cosmic_3window_uboone_15367667_4_gen_15367673_4\
_g4_15367682_4_detsim_tpc_15367771_4_detsim_optical_15367859_4_reco2D.root
```

It runs quite a number of reconstruction modules (13), plus some.

The format is:

```
TimeModule> run: 1 subRun: 5 event: 50 trackkalmanhit Track3DKalmanHit 105
```

meaning that the event #50 of run #1, subrun #5 has spent 105" (CPU time) running a `Track3DKalmanHit` module instance named `trackkalmanhit`.

art Timing service in action

(by `grep ^Time LogFile` uniq|)

```
TimeModule> run: 1 subRun: 5 event: 50 rns RandomNumberSaver 6.91414e-05
TimeModule> run: 1 subRun: 5 event: 50 trackkalmanhit Track3DKalmanHit 105
TimeModule> run: 1 subRun: 5 event: 50 trackkalmanhitcc Track3DKalmanHit 8
TimeModule> run: 1 subRun: 5 event: 50 spacepointfinder SpacePointFinder 2
TimeModule> run: 1 subRun: 5 event: 50 trackkalsps Track3DKalmanSPS 3.3585
TimeModule> run: 1 subRun: 5 event: 50 beziertracker BezierTrackerModule 3
TimeModule> run: 1 subRun: 5 event: 50 spacepointfindercc SpacePointFinder
TimeModule> run: 1 subRun: 5 event: 50 beziertrackercc BezierTrackerModule
TimeModule> run: 1 subRun: 5 event: 50 trackkalmanhitcalo Calorimetry 14.1
TimeModule> run: 1 subRun: 5 event: 50 trackkalmanhitcccalo Calorimetry 8.
TimeModule> run: 1 subRun: 5 event: 50 trackkalspscalo Calorimetry 6.37939
TimeModule> run: 1 subRun: 5 event: 50 beziertrackercccalo BezierCalorimet
TimeModule> run: 1 subRun: 5 event: 50 beziertrackercalo BezierCalorimetry
TimeModule> run: 1 subRun: 5 event: 50 beamflashcompat BeamFlashCompatibil
TimeModule> run: 1 subRun: 5 event: 50 TriggerResults TriggerResultInserte
TimeModule> run: 1 subRun: 5 event: 50 out1 RootOutput 0.583466
TimeEvent> run: 1 subRun: 5 event: 50 320.972
TimeReport ---------- Time  Summary ---[sec]----
TimeReport CPU = 2517.729662 Real = 2542.718000
TimeReport> Time report complete in 2529.08 seconds
```

art Timing service in action

This is well-formex text, which can be easily parsed. I have written my own (`SortModuleTimes.py LogFile`):

```
[basicstyle=\ttfamily]
RandomNumberSaver[rns]                           7.46965e-05" (RMS 20.3%)...
Track3DKalmanHit[trackkalmanhit]                 84.5906"     (RMS 35.2%)...
Track3DKalmanHit[trackkalmanhitcc]               63.3171"     (RMS 38.2%)...
SpacePointFinder[spacepointfinder]               11.6491"     (RMS 54.3%)...
Track3DKalmanSPS[trackkalsps]                    3.79274"     (RMS 35.6%)...
BezierTrackerModule[beziertracker]               29.1193"     (RMS 29.3%)...
SpacePointFinder[spacepointfindercc]             5.33569"     (RMS 51.1%)...
BezierTrackerModule[beziertrackercc]             22.72"       (RMS 43.5%)...
Calorimetry[trackkalmanhitcalo]                  13.1713"     (RMS 37.1%)...
Calorimetry[trackkalmanhitcccalo]                9.71266"     (RMS 34.5%)...
Calorimetry[trackkalspscalo]                     7.75189"     (RMS 49.4%)...
BezierCalorimetry[beziertrackercccalo]           0.00275586"  (RMS 21.8%)...
BezierCalorimetry[beziertrackercalo]             0.0031462"   (RMS 18.1%)...
BeamFlashCompatibilityCheck[beamflashcompat]     0.00579774"  (RMS 20.2%)...
TriggerResultInserter[TriggerResults]            3.03507e-05" (RMS  9.6%)...
RootOutput[out1]                                 0.599855"    (RMS 25.4%)...
=== events ===                                   252.207"     (RMS 34.9%)...
```

`Track3DKalmanHit` is our next stop...

Google `gperftools`

What `gperftools` does

`gperftools` provides statistical usage information for all the functions.
The job is sampled 100 times per second.

- + full call stack information reduces ambiguities
- + in-function timing
- + less than 2% overhead (i.e., slowdown)
- + conversion to `callgrind` format
- – sampling: information is subject to error
- – documentation is not very good; does it cope well with a busy system?
- – I couldn't actually extract the full stack information...

gperftools: usage

gperftools requires their library to be linked to the program to be profiled. The easiest way is to do it dynamically:

```
env LD_PRELOAD=libprofiler.so CPUPROFILE=gperftools.prof \
lar -c ./standard_reco_uboone_3D_cosmic_eric20140313.fcl \
  -s prodgenie_bnb_nu_cosmic_3window_uboone_15367667_4_gen_15367673_4\
_g4_15367682_4_detsim_tpc_15367771_4_detsim_optical_15367859_4_reco2D.root
```

or

```
larrun.sh --gperf \
  -s prodgenie_bnb_nu_cosmic_3window_uboone_15367667_4_gen_15367673_4\
_g4_15367682_4_detsim_tpc_15367771_4_detsim_optical_15367859_4_reco2D.root
  ./standard_reco_uboone_3D_cosmic_eric20140313.fcl
```

(larrun.sh is a helper script, a seldom-updated version available in LArSoft wiki profiling page)

gperftools in action

The output of gperftools is binary, and it can be converted in text or other formats by pprof script (provided with gperftools):

```
pprof --text lar gperftools.prof > gperftools.txt
```

The top of the output:

```
23681    9.3%   9.3%    23681    9.3% boost::numeric::ublas::basic_row_major
17295    6.8%  16.1%    17295    6.8% __log10_finite ??:?
16625    6.5%  22.6%    16625    6.5% std::local_Rb_tree_increment tree.cc:?
11845    4.6%  27.2%    11845    4.6% __exp_finite ??:?
 9677    3.8%  31.0%     9677    3.8% _IO_str_seekoff ??:?
 9462    3.7%  34.7%     9462    3.7% trkf::SpacePointAlg::makeSpacePoints@8
 6720    2.6%  37.3%    11757    4.6% trkf::SpacePointAlg::makeSpacePoints@8
```

We can see the number of samples we collected in each function, the fraction respect to the total, “exclusive” timing (and its cumulative value), the samples we collected in the functions or in the ones it called, and its fraction, “inclusive” timing.
Let’s sort after the latter (`--cum` option):

gperftools listing

The top of the output, skipping `main`-like functions:

```
    0   0.0%   0.0%   211908  83.0% main
[...]
    0   0.0%   0.0%   140808  55.2% trkf::Track3DKalmanHit::produce
    0   0.0%   0.0%    63622  24.9% trkf::SeedFinderAlgorithm::GetSeedsFro
  513   0.2%   0.2%    63584  24.9% trkf::SeedFinderAlgorithm::FindSeeds
19659   7.7%   7.9%    61726  24.2% trkf::SpacePointAlg::makeSpacePoints@8
   13   0.0%   7.9%    50961  20.0% matrix_assign (inline)
    8   0.0%   7.9%    45492  17.8% trkf::Propagator::noise_prop
   99   0.0%   7.9%    37397  14.6% trkf::KalmanFilterAlg::extendTrack
  155   0.1%   8.0%    36997  14.5% trkf::Propagator::vec_prop
   16   0.0%   8.0%    32677  12.8% trkf::Propagator::lin_prop
  572   0.2%   8.2%    30938  12.1% boost::numeric::ublas::indexing_matrix
[...]
    0   0.0%  19.9%    26636  10.4% trkf::BezierTrackerModule::produce
    4   0.0%  19.9%    26163  10.2% ServiceHandle (inline)
 2258   0.9%  20.7%    25812  10.1% apply (inline)
   12   0.0%  20.7%    25130   9.8% calo::Calorimetry::produce
  464   0.2%  20.9%    24273   9.5% boost::numeric::ublas::matrix_assign
  854   0.3%  21.3%    23172   9.1% trkf::BezierTrack::GetClosestApproach
[...]
    1   0.0%  30.2%    17290   6.8% trkf::BezierTrackerAlgorithm::FilterOv
   29   0.0%  30.2%    17287   6.8% trkf::KalmanFilterAlg::smoothTrack
```

A visualizer for `callgrind`

`gperftools` has a lot more of information, but it's not easy to get to it.

```
pprof -callgrind lar gperftools.prof > gperftools-callgrind.txt
```

converts the information in `callgrind`'s format. Why do we care?

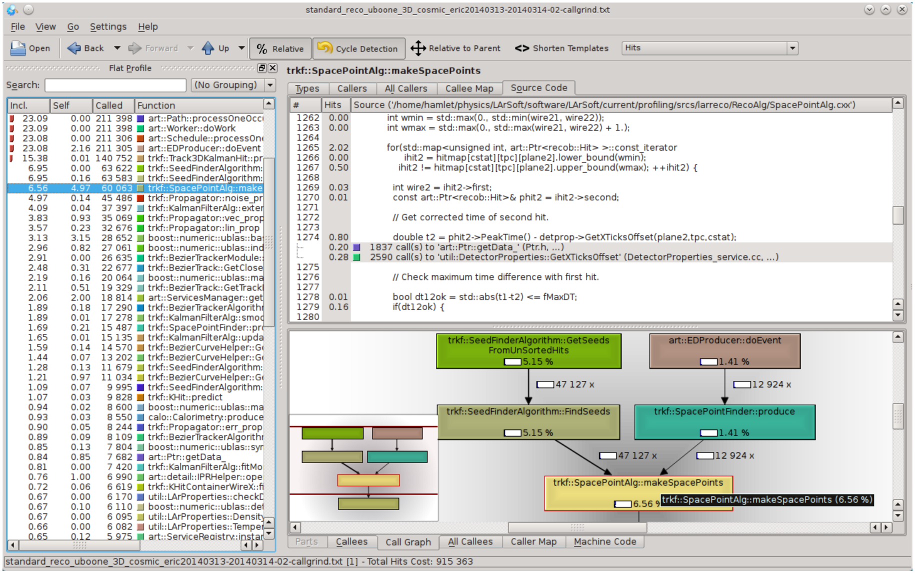
There are some graphic visualizer for callgrind!

- for KDE: `kcachegrind` (OK, it's for `cachegrind`... same format)
- in Eclipse's Linux Tools Project

Beware!!!

The conversion is far from being perfect, some values are wrong!

A visualizer for `callgrind` (II)



standard_reco_uboone_3D_cosmic_eric20140313-20140314-02-callgrind.txt
File View Go Settings Help
Open
Back
Forward
Up
Relative
Cycle Detection
Relative to Parent
Shorten Templates
Hits
Flat Profile
Search:
(No Grouping)
trkf::SpacePointAlg::makeSpacePoints
Types
Callers
All Callers
Callee Map
Source Code
trkf::SeedFinderAlgorithm::GetSeedsFromUnSortedHits
5.15 %
art::EDProducer::doEvent
1.41 %
47 127 x
12 924 x
trkf::SeedFinderAlgorithm::FindSeeds
5.15 %
trkf::SpacePointFinder::produce
1.41 %
47 127 x
12 924 x
trkf::SpacePointAlg::makeSpacePoints
6.56 %
trkf::SpacePointAlg::makeSpacePoints (6.56 %)
Parts
Callees
Call Graph
All Callees
Caller Map
Machine Code

Incl.	Self	Called	Function
23.09	0.00	211 398	art::Path::processOneOccu
23.09	0.00	211 398	art::Worker::doWork
23.08	0.00	211 306	art::Schedule::processOne
23.08	2.16	211 305	art::EDProducer::doEvent
15.38	0.01	140 752	trkf::Track3DKalmanHit::pr
6.95	0.00	63 622	trkf::SeedFinderAlgorithm:
6.95	0.16	63 583	trkf::SeedFinderAlgorithm:
6.56	4.97	60 063	trkf::SpacePointAlg::make
4.97	0.14	45 486	trkf::Propagator::noise_pr
4.09	0.04	37 397	trkf::KalmanFilterAlg::exter
3.83	0.93	35 069	trkf::Propagator::vec_prop
3.57	0.23	32 676	trkf::Propagator::lin_prop
3.13	3.15	28 652	boost::numeric::ublas::ba
2.96	0.82	27 061	boost::numeric::ublas::ind
2.91	0.00	26 635	trkf::BezierTrackerModule:
2.48	0.31	22 677	trkf::BezierTrack::GetClose
2.19	0.16	20 064	boost::numeric::ublas::ma
2.11	0.51	19 329	trkf::BezierTrack::GetTrack
2.06	2.00	18 814	art::ServicesManager::get
1.89	0.18	17 290	trkf::BezierTrackerAlgorith
1.89	0.01	17 278	trkf::KalmanFilterAlg::smoo
1.69	0.21	15 487	trkf::SpacePointFinder::pr
1.65	0.01	15 135	trkf::KalmanFilterAlg::upda
1.59	0.14	14 570	trkf::BezierCurveHelper::Ge
1.44	0.07	13 202	trkf::BezierCurveHelper::Ge
1.28	0.13	11 679	trkf::SeedFinderAlgorithm:
1.21	0.97	11 034	trkf::BezierCurveHelper::Ge
1.09	0.07	9 995	trkf::SeedFinderAlgorithm:
1.07	0.03	9 828	trkf::KHit::predict
0.94	0.02	8 600	boost::numeric::ublas::ma
0.93	0.03	8 550	calo::Calorimetry::produce
0.90	0.05	8 244	trkf::Propagator::err_prop
0.89	0.09	8 109	trkf::BezierTrackerAlgorith
0.85	0.13	7 804	boost::numeric::ublas::sym
0.84	0.85	7 682	art::Ptr::getData_
0.81	0.00	7 420	trkf::KalmanFilterAlg::fitMo
0.76	1.00	6 990	art::detail::IPRHelper::ope
0.72	0.06	6 619	trkf::KHitContainerWireX::fi
0.67	0.00	6 170	util::LArProperties::checkD
0.67	0.10	6 110	boost::numeric::ublas::det
0.67	0.00	6 095	util::LArProperties::Density
0.66	0.00	6 082	util::LArProperties::Temper
0.65	0.12	5 975	art::ServiceRegistry::instar

```
#     Hits  Source ('/home/hamlet/physics/LArSoft/software/LArSoft/current/profiling/srcs/larreco/RecoAlg/SpacePointAlg.cxx')
1262  0.00          int wmin = std::max(0., std::min(wire21, wire22));
1263  0.00          int wmax = std::max(0., std::max(wire21, wire22) + 1.);
1264
1265  2.02          for(std::map<unsigned int, art::Ptr<recob::Hit> >::const_iterator
1266  0.00                ihit2 = hitmap[cstat][tpc][plane2].lower_bound(wmin);
1267  0.50              ihit2 != hitmap[cstat][tpc][plane2].upper_bound(wmax); ++ihit2) {
1268
1269  0.03            int wire2 = ihit2->first;
1270  0.01            const art::Ptr<recob::Hit>& phit2 = ihit2->second;
1271
1272                  // Get corrected time of second hit.
1273
1274  0.80            double t2 = phit2->PeakTime() - detprop->GetXTicksOffset(plane2,tpc,cstat);
      0.20  1837 call(s) to 'art::Ptr::getData_' (Ptr.h, ...)
      0.28  2590 call(s) to 'util::DetectorProperties::GetXTicksOffset' (DetectorProperties_service.cc, ...)
1275
1276                  // Check maximum time difference with first hit.
1277
1278  0.01            bool dt12ok = std::abs(t1-t2) <= fMaxDT;
1279  0.16            if(dt12ok) {
1280
```

standard_reco_uboone_3D_cosmic_eric20140313-20140314-02-callgrind.txt [1] - Total Hits Cost: 915 363

valgrind

`valgrind` is *a sort of CPU emulator*, which dresses ("instruments") each instruction that the program runs.
The monitoring of the program is almost complete.
The price is an almost unbearable execution overhead.
On top of `valgrind`, a number of "tools" are available:

`memcheck`	memory leak detection (the "default" tool)
`callgrind`, `cachegrind`	CPU profiling
`massif`, `dhat`	memory profiling
`nulgrind`	"will run roughly 5 times more slowly than normal, for no useful effect"
...	and a few more

callgrind (valgrind tool)

What callgrind does

callgrind monitors *all* the function calls, counting how many times they happen and how long they take.

- + precise information: how many times each function has been called, how many CPU cycles it has used
- + in-function timing
- – information can be misleading (there no full stack information, the call graph is less informative than it looks)
- – valgrind-slow

callgrind (valgrind tool): usage

Run it straight with:

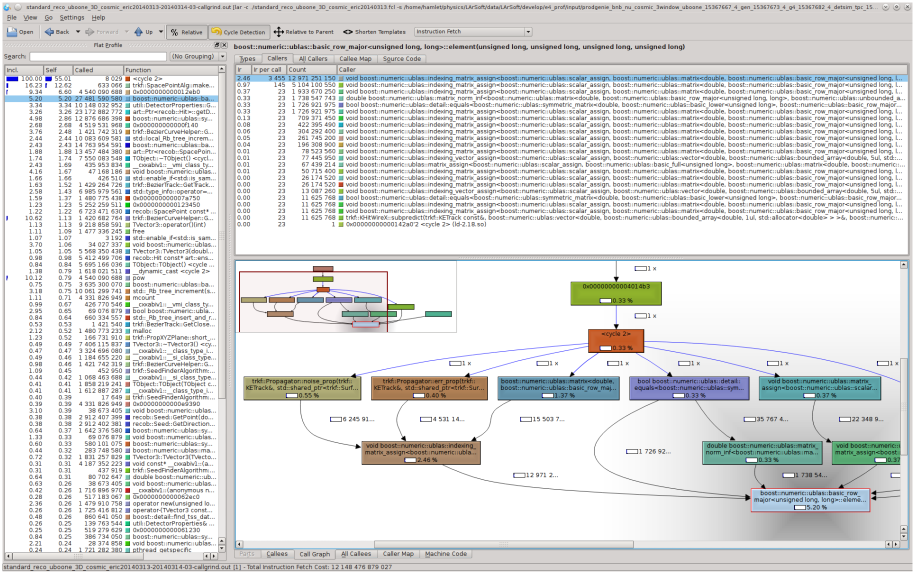
```
valgrind --tool=callgrind -n 5 \
lar -c ./standard_reco_uboone_3D_cosmic_eric20140313.fcl \
  -s prodgenie_bnb_nu_cosmic_3window_uboone_15367667_4_gen_15367673_4\
_g4_15367682_4_detsim_tpc_15367771_4_detsim_optical_15367859_4_reco2D.root
```

or

```
larrun.sh --callgrind \
  -s prodgenie_bnb_nu_cosmic_3window_uboone_15367667_4_gen_15367673_4\
_g4_15367682_4_detsim_tpc_15367771_4_detsim_optical_15367859_4_reco2D.root
  ./standard_reco_uboone_3D_cosmic_eric20140313.fcl -n 5
```

Then do something else.

`callgrind` (`valgrind` tool) in action (I)



standard_reco_uboone_3D_cosmic_eric20140313-20140314-03-callgrind.out [lar -c ./standard_reco_uboone_3D_cosmic_eric20140313.fcl -s /home/hamlet/physics/LArSoft/data/LArSoft/develop/e4_prof/input/prodgenie_bnb_nu_cosmic_3window_uboone_15367667_4_gen_15367673_4_g4_15367682_4_detsim_tpc_15...

File View Go Settings Help

Open | Back | Forward | Up | Relative | Cycle Detection | Relative to Parent | Shorten Templates | Instruction Fetch

Flat Profile

Search: (No Grouping)

Incl.	Self	Called	Function
100.00	55.01	8 029	<cycle 2>
16.23	12.62	633 066	trkf::SpacePointAlg::make...
9.34	6.60	4 540 090 688	0x000000000012eb0
5.20	5.20	27 481 590 580	boost::numeric::ublas::ba...
3.34	3.34	10 148 032 952	util::DetectorProperties::G...
3.26	3.26	23 172 882 772	art::Ptr<recob::Hit>::getD...
4.98	2.86	12 876 686 398	boost::numeric::ublas::sy...
2.68	2.68	4 519 531 968	0x00000000000f140
3.76	2.48	1 421 742 319	trkf::BezierCurveHelper::G...
2.44	2.44	10 083 609 581	std::local_Rb_tree_increm...
2.43	2.43	14 763 954 591	boost::numeric::ublas::ba...
1.88	1.88	13 457 484 380	art::Ptr<recob::SpacePoin...
1.74	1.74	7 550 083 548	TObject::~TObject() <cycl...
2.43	1.69	435 953 834	__cxxabiv1::__vmi_class_ty...
4.16	1.67	47 168 186	void boost::numeric::ublas...
1.66	1.66	426 510	std::enable_if<std::is_sam...
1.63	1.52	1 429 264 726	trkf::BezierTrack::GetTrack...
2.58	1.43	6 985 979 561	std::type_info::operator=...
1.59	1.37	1 480 775 438	0x00000000007a750
1.23	1.23	5 252 259 511	0x000000000123450
1.22	1.22	6 723 471 630	recob::SpacePoint const* ...
10.62	1.13	1 420 682 764	trkf::BezierCurveHelper::G...
1.13	1.13	9 218 858 591	TVector3::operator()(int)
1.11	1.09	1 477 336 245	free
1.07	1.07	3 192	std::enable_if<std::is_sam...
3.70	1.06	34 027 337	void boost::numeric::ublas...
1.05	1.05	5 568 350 438	TVector3::TVector3(doubl...
0.98	0.98	5 412 499 706	recob::Hit const* art::ens...
0.84	0.84	5 695 166 036	TObject::TObject() <cycle ...
1.38	0.79	1 618 021 511	__dynamic_cast <cycle 2>
10.12	0.79	4 540 090 688	pow
0.75	0.75	3 635 300 070	boost::numeric::ublas::ba...
3.18	0.75	10 061 299 741	std::_Rb_tree_increment(s...
1.11	0.71	4 331 826 949	mcount
0.99	0.67	426 770 546	__cxxabiv1::__vmi_class_ty...
2.95	0.65	69 076 879	bool boost::numeric::ubla...
0.84	0.64	660 334 557	std::_Rb_tree_insert_and_r...
0.53	0.53	1 421 540	trkf::BezierTrack::GetClose...
2.12	0.52	1 480 773 233	malloc
1.23	0.52	166 731 910	trkf::PropXYZPlane::short_...
0.49	0.49	7 406 115 837	TVector3::~TVector3() <cy...
0.47	0.47	3 324 696 080	__cxxabiv1::__class_type_i...
0.49	0.46	1 184 655 220	__cxxabiv1::__si_class_type...
0.98	0.46	1 421 742 319	trkf::BezierCurveHelper::G...
1.09	0.45	452 950	trkf::SeedFinderAlgorithm:...
0.44	0.42	1 068 463 688	__cxxabiv1::__si_class_type...
0.41	0.41	1 858 219 241	TObject::TObject(TObject c...
0.41	0.41	1 612 887 287	__cxxabiv1::__class_type_i...
0.40	0.39	17 649	trkf::SeedFinderAlgorithm:...
0.39	0.39	4 331 826 949	0x0000000000e9390
3.10	0.39	38 673 405	void boost::numeric::ublas...
0.38	0.38	2 912 407 399	recob::Seed::GetPoint(do...
0.38	0.38	2 912 402 381	recob::Seed::GetDirection...
0.64	0.37	1 642 376 580	boost::numeric::ublas::sy...
1.33	0.33	69 076 879	void boost::numeric::ublas...
0.60	0.33	580 101 075	boost::numeric::ublas::sy...
0.44	0.32	283 748 580	boost::numeric::ublas::ma...
0.72	0.32	1 831 257 829	TVector3::TVector3(TVecto...
0.31	0.31	4 187 352 223	void const* __cxxabiv1::(a...
0.31	0.31	437 919	trkf::SeedFinderAlgorithm:...
0.64	0.31	80 702 647	double boost::numeric::ub...
0.63	0.26	38 673 405	void boost::numeric::ublas...
0.42	0.26	1 716 896 970	__cxxabiv1::(anonymous n...
0.28	0.26	517 183 067	0x000000000062ec0
2.36	0.26	1 479 910 758	operator new(unsigned lo...
0.26	0.26	1 725 416 812	operator-(TVector3 const...
0.48	0.26	860 641 050	boost::detail::find_tss_dat...
0.26	0.25	139 763 544	util::DetectorProperties& ...
0.25	0.25	519 279 629	0x000000000061230
0.84	0.25	386 734 050	boost::numeric::ublas::sy...
2.21	0.24	28 374 858	void boost::numeric::ublas...
0.24	0.24	1 721 282 380	pthread_getspecific

boost::numeric::ublas::basic_row_major<unsigned long, long>::element(unsigned long, unsigned long, unsigned long, unsigned long)

Types | Callers | All Callers | Callee Map | Source Code

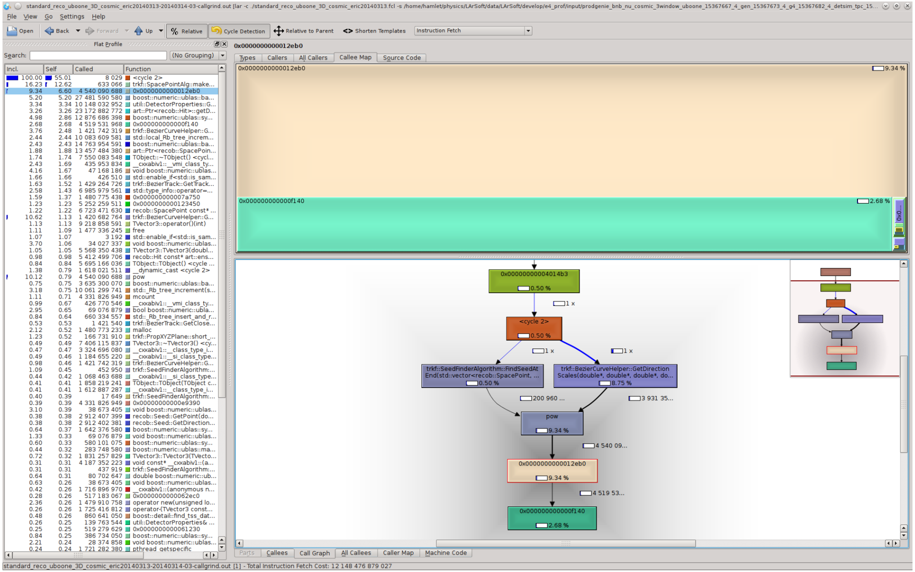
Ir	Ir per call	Count	Caller
2.46	3 455	12 971 251 150	void boost::numeric::ublas::indexing_matrix_assign<boost::numeric::ublas::scalar_assign, boost::numeric::ublas::matrix<double, boost::numeric::ublas::basic_row_major<unsigned long, l...
0.97	145	5 104 100 550	void boost::numeric::ublas::indexing_matrix_assign<boost::numeric::ublas::scalar_assign, boost::numeric::ublas::matrix<double, boost::numeric::ublas::basic_row_major<unsigned long, l...
0.37	23	1 933 670 250	void boost::numeric::ublas::indexing_matrix_assign<boost::numeric::ublas::scalar_assign, boost::numeric::ublas::matrix<double, boost::numeric::ublas::basic_row_major<unsigned long, l...
0.33	23	1 738 547 743	double boost::numeric::ublas::matrix_norm_inf<boost::numeric::ublas::matrix<double, boost::numeric::ublas::basic_row_major<unsigned long, long>, boost::numeric::ublas::unbounded_a...
0.33	23	1 726 921 975	bool boost::numeric::ublas::detail::equals<boost::numeric::ublas::symmetric_matrix<double, boost::numeric::ublas::basic_lower<unsigned long>, boost::numeric::ublas::basic_row_major...
0.33	23	1 726 921 975	void boost::numeric::ublas::indexing_matrix_assign<boost::numeric::ublas::scalar_assign, boost::numeric::ublas::matrix<double, boost::numeric::ublas::basic_row_major<unsigned long, l...
0.13	23	709 371 450	void boost::numeric::ublas::indexing_matrix_assign<boost::numeric::ublas::scalar_assign, boost::numeric::ublas::matrix<double, boost::numeric::ublas::basic_row_major<unsigned long, l...
0.08	23	422 395 490	void boost::numeric::ublas::indexing_matrix_assign<boost::numeric::ublas::scalar_assign, boost::numeric::ublas::matrix<double, boost::numeric::ublas::basic_row_major<unsigned long, l...
0.06	23	304 292 400	void boost::numeric::ublas::indexing_matrix_assign<boost::numeric::ublas::scalar_assign, boost::numeric::ublas::matrix<double, boost::numeric::ublas::basic_row_major<unsigned long, l...
0.05	23	261 745 200	void boost::numeric::ublas::indexing_matrix_assign<boost::numeric::ublas::scalar_assign, boost::numeric::ublas::matrix<double, boost::numeric::ublas::basic_row_major<unsigned long, l...
0.04	23	196 308 900	void boost::numeric::ublas::indexing_matrix_assign<boost::numeric::ublas::scalar_assign, boost::numeric::ublas::matrix<double, boost::numeric::ublas::basic_row_major<unsigned long, l...
0.01	23	78 523 560	void boost::numeric::ublas::indexing_matrix_assign<boost::numeric::ublas::scalar_assign, boost::numeric::ublas::matrix<double, boost::numeric::ublas::basic_row_major<unsigned long, l...
0.01	23	77 445 950	void boost::numeric::ublas::indexing_vector_assign<boost::numeric::ublas::scalar_assign, boost::numeric::ublas::vector<double, boost::numeric::ublas::bounded_array<double, 5ul, std::...
0.01	23	67 439 214	void boost::numeric::ublas::matrix_assign<boost::numeric::ublas::scalar_assign, boost::numeric::ublas::basic_full<unsigned long>, boost::numeric::ublas::matrix<double, boost::numeric::...
0.01	23	50 715 400	void boost::numeric::ublas::indexing_matrix_assign<boost::numeric::ublas::scalar_assign, boost::numeric::ublas::matrix<double, boost::numeric::ublas::basic_row_major<unsigned long, l...
0.00	23	26 174 520	void boost::numeric::ublas::indexing_matrix_assign<boost::numeric::ublas::scalar_assign, boost::numeric::ublas::matrix<double, boost::numeric::ublas::basic_row_major<unsigned long, l...
0.00	23	26 174 520	void boost::numeric::ublas::indexing_matrix_assign<boost::numeric::ublas::scalar_assign, boost::numeric::ublas::matrix<double, boost::numeric::ublas::basic_row_major<unsigned long, l...
0.00	23	13 087 260	void boost::numeric::ublas::indexing_vector_assign<boost::numeric::ublas::scalar_assign, boost::numeric::ublas::vector<double, boost::numeric::ublas::bounded_array<double, 5ul, std::...
0.00	23	11 625 768	bool boost::numeric::ublas::detail::equals<boost::numeric::ublas::symmetric_matrix<double, boost::numeric::ublas::basic_lower<unsigned long>, boost::numeric::ublas::basic_row_major...
0.00	23	11 625 768	void boost::numeric::ublas::indexing_matrix_assign<boost::numeric::ublas::scalar_assign, boost::numeric::ublas::matrix<double, boost::numeric::ublas::basic_row_major<unsigned long, l...
0.00	23	11 625 768	void boost::numeric::ublas::indexing_matrix_assign<boost::numeric::ublas::scalar_assign, boost::numeric::ublas::matrix<double, boost::numeric::ublas::basic_row_major<unsigned long, l...
0.00	23	11 625 768	trkf::KHitWireX::subpredict(trkf::KETrack const&, boost::numeric::ublas::vector<double, boost::numeric::ublas::bounded_array<double, 1ul, std::allocator<double> > >&, boost::numeric::...
0.00	23	1	0x00000000000142a0'2 <cycle 2> (ld-2.18.so)

1 x
0x00000000004014b3
0.33 %
1 x
<cycle 2>
0.33 %
1 x
1 x
1 x
1 x
1 x
1 x
trkf::Propagator::noise_prop(trkf::KETrack&, std::shared_ptr<trkf::Surf...
0.55 %
trkf::Propagator::err_prop(trkf::KETrack&, std::shared_ptr<trkf::Sur...
0.40 %
boost::numeric::ublas::matrix<double, boost::numeric::ublas::basic_row_maj...
1.37 %
bool boost::numeric::ublas::detail::equals<boost::numeric::ublas::sym...
0.33 %
void boost::numeric::ublas::matrix_assign<boost::numeric::ublas::scalar...
0.37 %
6 245 91...
4 531 14...
15 503 7...
35 767 4...
22 348 9...
void boost::numeric::ublas::indexing_matrix_assign<boost::numeric::ubla...
2.46 %
1 726 92...
double boost::numeric::ublas::matrix_norm_inf<boost::numeric::ublas::ma...
0.33 %
void boost::numeric::matrix_assign<boost::
0.37
12 971 2...
1 738 54...
boost::numeric::ublas::basic_row_major<unsigned long, long>::eleme...
5.20 %

Parts | Callees | Call Graph | All Callees | Caller Map | Machine Code

standard_reco_uboone_3D_cosmic_eric20140313-20140314-03-callgrind.out [1] - Total Instruction Fetch Cost: 12 148 476 879 027

`callgrind` (`valgrind` tool) in action (II)



standard_reco_uboone_3D_cosmic_eric20140313-20140314-03-callgrind.out [lar -c ./standard_reco_uboone_3D_cosmic_eric20140313.fcl -s /home/hamlet/physics/LArSoft/data/LArSoft/develop/e4_prof/input/prodgenie_bnb_nu_cosmic_3window_uboone_15367667_4_gen_15367673_4_g4_15367682_4_detsim_tpc_15...

Incl.	Self	Called	Function
100.00	55.01	8 029	<cycle 2>
16.23	12.62	633 066	trkf::SpacePointAlg::make...
9.34	6.60	4 540 090 688	0x0000000000012eb0
5.20	5.20	27 481 590 580	boost::numeric::ublas::ba...
3.34	3.34	10 148 032 952	util::DetectorProperties::G...
3.26	3.26	23 172 882 772	art::Ptr<recob::Hit>::getD...
4.98	2.86	12 876 686 398	boost::numeric::ublas::sy...
2.68	2.68	4 519 531 968	0x000000000000f140
3.76	2.48	1 421 742 319	trkf::BezierCurveHelper::G...
2.44	2.44	10 083 609 581	std::local_Rb_tree_increm...
2.43	2.43	14 763 954 591	boost::numeric::ublas::ba...
1.88	1.88	13 457 484 380	art::Ptr<recob::SpacePoin...
1.74	1.74	7 550 083 548	TObject::~TObject() <cycl...
2.43	1.69	435 953 834	__cxxabiv1::__vmi_class_ty...
4.16	1.67	47 168 186	void boost::numeric::ublas...
1.66	1.66	426 510	std::enable_if<std::is_sam...
1.63	1.52	1 429 264 726	trkf::BezierTrack::GetTrack...
2.58	1.43	6 985 979 561	std::type_info::operator=...
1.59	1.37	1 480 775 438	0x000000000007a750
1.23	1.23	5 252 259 511	0x0000000000123450
1.22	1.22	6 723 471 630	recob::SpacePoint const* ...
10.62	1.13	1 420 682 764	trkf::BezierCurveHelper::G...
1.13	1.13	9 218 858 591	TVector3::operator()(int)
1.11	1.09	1 477 336 245	free
1.07	1.07	3 192	std::enable_if<std::is_sam...
3.70	1.06	34 027 337	void boost::numeric::ublas...
1.05	1.05	5 568 350 438	TVector3::TVector3(doubl...
0.98	0.98	5 412 499 706	recob::Hit const* art::ens...
0.84	0.84	5 695 166 036	TObject::TObject() <cycle ...
1.38	0.79	1 618 021 511	__dynamic_cast <cycle 2>
10.12	0.79	4 540 090 688	pow
0.75	0.75	3 635 300 070	boost::numeric::ublas::ba...
3.18	0.75	10 061 299 741	std::_Rb_tree_increment(s...
1.11	0.71	4 331 826 949	mcount
0.99	0.67	426 770 546	__cxxabiv1::__vmi_class_ty...
2.95	0.65	69 076 879	bool boost::numeric::ubla...
0.84	0.64	660 334 557	std::_Rb_tree_insert_and_r...
0.53	0.53	1 421 540	trkf::BezierTrack::GetClose...
2.12	0.52	1 480 773 233	malloc
1.23	0.52	166 731 910	trkf::PropXYZPlane::short_...
0.49	0.49	7 406 115 837	TVector3::~TVector3() <cy...
0.47	0.47	3 324 696 080	__cxxabiv1::__class_type_i...
0.49	0.46	1 184 655 220	__cxxabiv1::__si_class_type...
0.98	0.46	1 421 742 319	trkf::BezierCurveHelper::G...
1.09	0.45	452 950	trkf::SeedFinderAlgorithm:...
0.44	0.42	1 068 463 688	__cxxabiv1::__si_class_type...
0.41	0.41	1 858 219 241	TObject::TObject(TObject c...
0.41	0.41	1 612 887 287	__cxxabiv1::__class_type_i...
0.40	0.39	17 649	trkf::SeedFinderAlgorithm:...
0.39	0.39	4 331 826 949	0x00000000000e9390
3.10	0.39	38 673 405	void boost::numeric::ublas...
0.38	0.38	2 912 407 399	recob::Seed::GetPoint(do...
0.38	0.38	2 912 402 381	recob::Seed::GetDirection...
0.64	0.37	1 642 376 580	boost::numeric::ublas::sy...
1.33	0.33	69 076 879	void boost::numeric::ublas...
0.60	0.33	580 101 075	boost::numeric::ublas::sy...
0.44	0.32	283 748 580	boost::numeric::ublas::ma...
0.72	0.32	1 831 257 829	TVector3::TVector3(TVecto...
0.31	0.31	4 187 352 223	void const* __cxxabiv1::(a...
0.31	0.31	437 919	trkf::SeedFinderAlgorithm:...
0.64	0.31	80 702 647	double boost::numeric::ub...
0.63	0.26	38 673 405	void boost::numeric::ublas...
0.42	0.26	1 716 896 970	__cxxabiv1::(anonymous n...
0.28	0.26	517 183 067	0x0000000000062ec0
2.36	0.26	1 479 910 758	operator new(unsigned lo...
0.26	0.26	1 725 416 812	operator-(TVector3 const...
0.48	0.26	860 641 050	boost::detail::find_tss_dat...
0.26	0.25	139 763 544	util::DetectorProperties& ...
0.25	0.25	519 279 629	0x0000000000061230
0.84	0.25	386 734 050	boost::numeric::ublas::sy...
2.21	0.24	28 374 858	void boost::numeric::ublas...
0.24	0.24	1 721 282 380	pthread_getspecific

standard_reco_uboone_3D_cosmic_eric20140313-20140314-03-callgrind.out [1] - Total Instruction Fetch Cost: 12 148 476 879 027

CPU profiling summary

- tools provide complementary information
- tools information can be deceiving (or plainly wrong)
- once the bottleneck is detected, still work to be done!

Grampa’s hints:

- beware of loops with a lot of iterations
  - put only what strictly needed, pull the rest outside
  - dont’ write messages there (even `mf::LogDebug()` ones, since the message is still processed)

Example (I)

This is where `gperftools` is pointing to a nested `for` loop:
`larreco/RecoAlg/SpacePointAlg.cxx`, line 1267

```
for(std::map<unsigned int, art::Ptr<recob::Hit> >::const_iterator
      ihit2 = hitmap[cstat][tpc][plane2].lower_bound(wmin);
    ihit2 != hitmap[cstat][tpc][plane2].upper_bound(wmax); ++ihit2) {

  int wire2 = ihit2->first;
  const art::Ptr<recob::Hit>& phit2 = ihit2->second;

  // Get corrected time of second hit.

  double t2 = phit2->PeakTime() - detprop->GetXTicksOffset(plane2,tpc,csta
```

Example (II)

Pulling out stuff from the loop:

```
const double TicksOffset2 = detprop->GetXTicksOffset(plane2,tpc,cstat);

std::map<unsigned int, art::Ptr<recob::Hit> >::const_iterator
  ihit2 = hitmap[cstat][tpc][plane2].lower_bound(wmin),
  ihit2end = hitmap[cstat][tpc][plane2].upper_bound(wmax);
for(; ihit2 != ihit2end; ++ihit2) {

  int wire2 = ihit2->first;
  const art::Ptr<recob::Hit>& phit2 = ihit2->second;

  // Get corrected time of second hit.

  double t2 = phit2->PeakTime() - TicksOffset2;
```

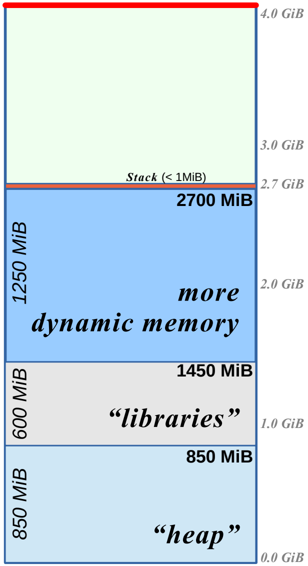
might help (or not).
In this case, acting in this way in 4 different points of the code, `SpacePointFinder` became 20% faster and `Track3DKalmanHit` 15%, while the other modules fluctuated on $\pm 2\%$. On the event the gain was 10%.

A map of the memory

The memory “footprint” of a program is how much memory the program needs in order to run.
We start to be interested to it when remote sites kill our job based on such a figure, for example 4 GB,. The memory is made of:

libraries	(mostly) not our fault, but it’s accounted on us nevertheless
stacks	where the local variables stay; usually negligible
heap & Co.	where the dynamic memory is allocated (via `new`)

The total of this memory is known as *virtual size* (VSIZE). The resident size (RSS) excludes the libraries (and something more).



Memory usage profiling

Goal:

Map the memory usage to the places where it is requested, and analyse its evolution in time.

Memory usage needs to be controlled in environments with limited resources (e.g., all the CPU farms).

- large pools of memory might be reduced by on-demand loading
- redesign of algorithms can reduce the need for caching (e.g., convert from one loop to load everything and one to analyse to one loop to load and analyse each event)
- constant increase of memory without its release can be identified and fixed
- sudden peaks of memory usage can be smoothed
- memory fragmentation can be mitigated

Memory profiling tools

`pmap`	(Linux command): total memory map
`SimpleMemoryCheck`	(`art` service): monitor per-event, module-level memory increase
`massif`	(`valgrind` tool) records each allocation and deallocation

There are, as usual, more tools (for example, other `valgrind` tools, `memcheck` and `dhat`, and `gperftools` also has memory check capabilities),

Linux memory map (pmap)

Linux knows how much memory it is allowing the programs to run. This information can be accessed by:

from `procfs` `/proc/PID/maps` "file" gives a picture of the memory used by process number *PID*

`AddMemoryMap.py` (a script of mine) does the same (more control)

`pmap` has different options for the same thing

If you are running MacOS, there must be other ways (which I ignore).

Out of `AddMemoryMap.py -s mapsize /proc/PID/maps`:

```
    [...]
   676212 KiB    25012 KiB | /usr/lib64/libicudata.so.52.1
  1523792 KiB   847580 KiB | [heap]
  2794028 KiB  1270236 KiB |
/proc/PID/maps: 2861084672 bytes (2728.54 MiB) in 1099 pages and 250 group
```

with the first column the total so far, the dynamic memory shown in blue and the total memory usage in red.

art SimpleMemoryCheck service

What SimpleMemoryCheck service does

At the end of each art *module*, if the memory usage is increased, that's printed out.
The reported memory is both "VSIZE" and "RSS".
A summary at the end also shows peak memory usage.

This tool is good to detect steadily increase on memory and a first guess of where it happens.

+ out of the box with art
+ most of MicroBooNE FCL files already have it enabled
+ if not, just add `services.SimpleMemoryCheck: {}` to the configuration

– rough code identification: just tells the module
– rough memory estimation: only samples module boundaries
– reported peak memory can be far from the actual one

art SimpleMemoryCheck in action

From `grep Memory LogFile.log` of a 10-event run log:

```
%MSG-w MemoryCheck:  Track3DKalmanHit:trackkalmanhit 14-Mar-2014 18:32:43
run: 1 subRun: 5 event: 42
MemoryCheck: module Track3DKalmanHit:trackkalmanhit VSIZE 1819.14 58.4375
%MSG-w MemoryCheck:  Track3DKalmanHit:trackkalmanhitcc 14-Mar-2014 18:34:1
run: 1 subRun: 5 event: 42
MemoryCheck: module Track3DKalmanHit:trackkalmanhitcc VSIZE 1831.12 11.984
%MSG-w MemoryCheck:  SpacePointFinder:spacepointfinder 14-Mar-2014 18:34:2
run: 1 subRun: 5 event: 42
MemoryCheck: module SpacePointFinder:spacepointfinder VSIZE 1831.96 0.8398
%MSG-w MemoryCheck:  Track3DKalmanSPS:trackkalsps 14-Mar-2014 18:34:34 CDT
run: 1 subRun: 5 event: 42
MemoryCheck: module Track3DKalmanSPS:trackkalsps VSIZE 1839.23 7.26172 RSS
[...]
%MSG-w MemoryCheck:  Calorimetry:trackkalspscalo 14-Mar-2014 18:53:50 CDT
run: 1 subRun: 5 event: 46
MemoryCheck: module Calorimetry:trackkalspscalo VSIZE 1888.11 5.03516 RSS
%MSG-w MemoryCheck:  RootOutput:out1 14-Mar-2014 18:53:50 CDT  run: 1 subR
MemoryCheck: module RootOutput:out1 VSIZE 1905.07 16.9688 RSS 1232.01 18.5
MemoryReport> Peak virtual size 1905.07 Mbytes
```

The peak memory can be wrong, but it's still informative.

massif (valgrind tool)

What massif does

massif monitors *all* the memory allocations, and where they occur. It samples frequently, but reports details only when the memory changes dramatically.

- + precise information: how much memory allocated for each function in a call stack
- + possibility to track memory at lower level (mmap instead of malloc/new)
- + possibility to track stacks
- + graphic visualizers help
- – detailed sampling can miss the part you are interested in
- – valgrind-slow; tracking stacks is valgrind^2-slow

massif (valgrind tool): usage

Run it straight with:

```
valgrind --tool=massif -n 5 \
lar -c ./standard_reco_uboone_3D_cosmic_eric20140313.fcl \
  -s prodgenie_bnb_nu_cosmic_3window_uboone_15367667_4_gen_15367673_4\
_g4_15367682_4_detsim_tpc_15367771_4_detsim_optical_15367859_4_reco2D.root
```

or

```
larrun.sh --massif \
  -s prodgenie_bnb_nu_cosmic_3window_uboone_15367667_4_gen_15367673_4\
_g4_15367682_4_detsim_tpc_15367771_4_detsim_optical_15367859_4_reco2D.root
  ./standard_reco_uboone_3D_cosmic_eric20140313.fcl -n 5
```

Then do something else. Chances are that you find the problem by looking at the code before massif is done.

massif (valgrind tool): example

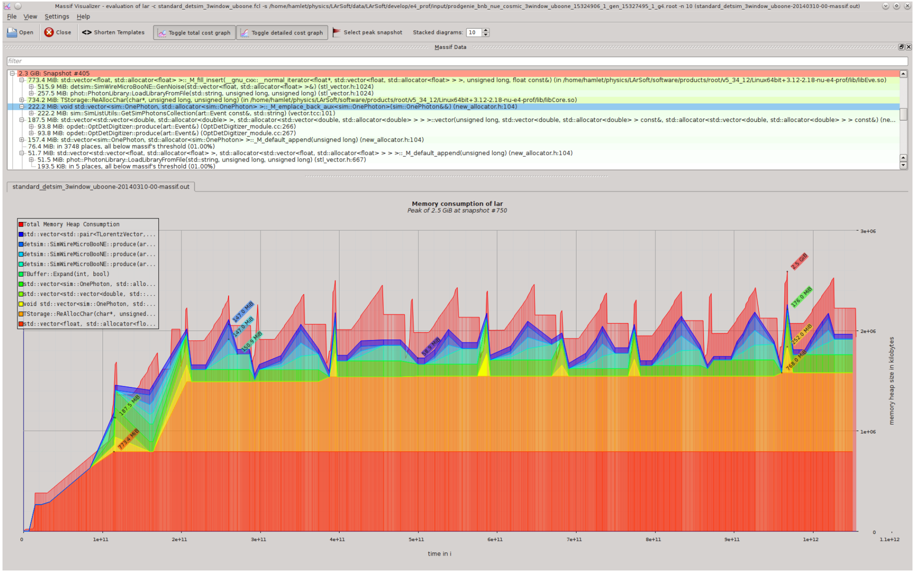
A detailed snapshot of a 2-event reco run writing a large event:

```
snapshot=951
#-----------
time=2755749742080
mem_heap_B=2222708828
mem_heap_extra_B=47481604
...
n10: 2222708828 (heap allocation functions) malloc/new/new[], --alloc-fns, etc.
 n2: 1244677004 ...: TStorage::ReAllocChar(...) (in .../libCore.so)
  n4: 631258836 ...: TBuffer::Expand(int, bool) (in .../libCore.so)
   n1: 433599113 ...: TBasket::ReadBasketBuffers(long long, int, TFile*) (in .../libTree.so)
    ...
              n1: 166661418 ...: art::Ptr<raw::RawDigit>::getData_() const (.../Ptr.h:457)
               n1: 166661418 ...: trkf::SeedFinderAlgorithm::FindSeeds() (.../Ptr.h:344)
                n1: 166661418 ...: trkf::SeedFinderAlgorithm::GetSeedsFromUnSortedHits(...) (.../S
                 n1: 166661418 ...: trkf::Track3DKalmanHit::produce(art::Event&) (.../Track3DKalma
                  ...
   n1: 166435561 ...: TBasket::WriteBuffer() (in libTree.so)
    ...
               n1: 166435561 ...: art::RootOutput::write(...) (.../RootOutput_module.cc:155)
```

(ellipses are mine)

It says 2.2 GB of memory are allocated via `malloc`/`new`, more than half by ROOT tree (a good lot for writing, plus some for reading).

massif example: general shape (detsim)



Massif Visualizer - evaluation of lar -c standard_detsim_3window_uboone.fcl -s /home/hamlet/physics/LArSoft/data/LArSoft/develop/e4_prof/input/prodgenie_bnb_nue_cosmic_3window_uboone_15324906_1_gen_15327495_1_g4.root -n 10 (standard_detsim_3window_uboone-20140310-00-massif.out)

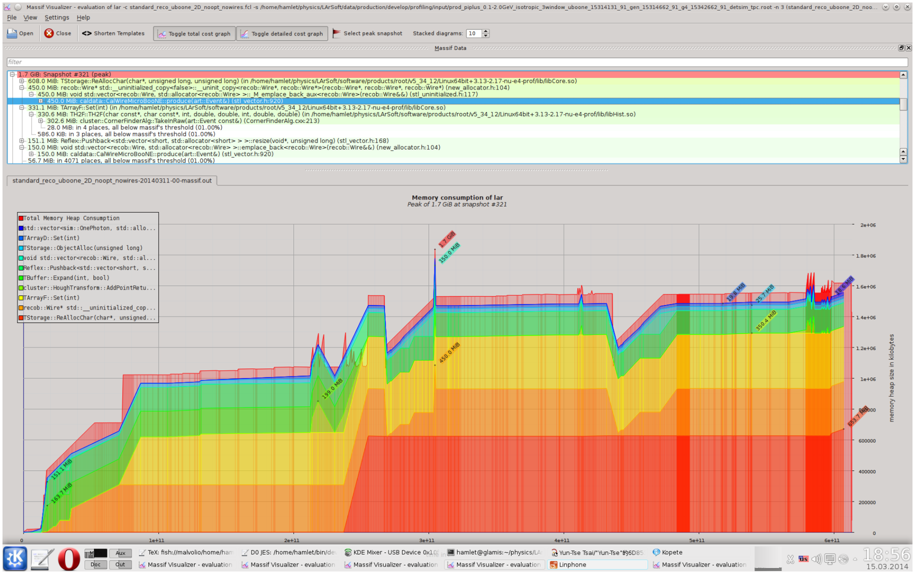
Stacked diagrams: 10

```
2.3 GiB: Snapshot #405
  773.4 MiB: std::vector<float, std::allocator<float> >::_M_fill_insert(__gnu_cxx::__normal_iterator<float*, std::vector<float, std::allocator<float> > >, unsigned long, float const&) (in /home/hamlet/physics/LArSoft/software/products/root/v5_34_12/Linux64bit+3.12-2.18-nu-e4-prof/lib/libEve.so)
    515.9 MiB: detsim::SimWireMicroBooNE::GenNoise(std::vector<float, std::allocator<float> >&) (stl_vector.h:1024)
    257.5 MiB: phot::PhotonLibrary::LoadLibraryFromFile(std::string, unsigned long, unsigned long) (stl_vector.h:1024)
  734.2 MiB: TStorage::ReAllocChar(char*, unsigned long, unsigned long) (in /home/hamlet/physics/LArSoft/software/products/root/v5_34_12/Linux64bit+3.12-2.18-nu-e4-prof/lib/libCore.so)
  222.2 MiB: void std::vector<sim::OnePhoton, std::allocator<sim::OnePhoton> >::_M_emplace_back_aux<sim::OnePhoton>(sim::OnePhoton&&) (new_allocator.h:104)
    222.2 MiB: sim::SimListUtils::GetSimPhotonsCollection(art::Event const&, std::string) (vector.tcc:101)
  187.5 MiB: std::vector<std::vector<double, std::allocator<double> >, std::allocator<std::vector<double, std::allocator<double> > > >::vector(unsigned long, std::vector<double, std::allocator<double> > const&, std::allocator<std::vector<double, std::allocator<double> > > const&) (ne...
    93.8 MiB: opdet::OptDetDigitizer::produce(art::Event&) (OptDetDigitizer_module.cc:266)
    93.8 MiB: opdet::OptDetDigitizer::produce(art::Event&) (OptDetDigitizer_module.cc:267)
  157.4 MiB: std::vector<sim::OnePhoton, std::allocator<sim::OnePhoton> >::_M_default_append(unsigned long) (new_allocator.h:104)
  76.4 MiB: in 3748 places, all below massif's threshold (01.00%)
  51.7 MiB: std::vector<std::vector<float, std::allocator<float> >, std::allocator<std::vector<float, std::allocator<float> > > >::_M_default_append(unsigned long) (new_allocator.h:104)
    51.5 MiB: phot::PhotonLibrary::LoadLibraryFromFile(std::string, unsigned long, unsigned long) (stl_vector.h:667)
    193.5 KiB: in 5 places, all below massif's threshold (01.00%)
```

standard_detsim_3window_uboone-20140310-00-massif.out

File
View
Settings
Help
Open
Close
Shorten Templates
Toggle total cost graph
Toggle detailed cost graph
Select peak snapshot
Massif Data
filter
Memory consumption of lar
Peak of 2.5 GiB at snapshot #750
Total Memory Heap Consumption
std::vector<std::pair<TLorentzVector,...
detsim::SimWireMicroBooNE::produce(ar...
detsim::SimWireMicroBooNE::produce(ar...
detsim::SimWireMicroBooNE::produce(ar...
TBuffer::Expand(int, bool)
std::vector<sim::OnePhoton, std::allo...
std::vector<std::vector<double, std::...
void std::vector<sim::OnePhoton, std:...
TStorage::ReAllocChar(char*, unsigned...
std::vector<float, std::allocator<flo...
2.5 GiB
176.0 MiB
252.0 MiB
768.0 MiB
147.0 MiB
147.0 MiB
150.9 MiB
58.9 MiB
187.5 MiB
773.4 MiB
3e+06
2e+06
1e+06
0
memory heap size in kilobytes
0
1e+11
2e+11
3e+11
4e+11
5e+11
6e+11
7e+11
8e+11
9e+11
1e+12
1.1e+12
time in i

massif example: memory leak + spike (reco 2D)



```
Massif Visualizer - evaluation of lar -c standard_reco_uboone_2D_noopt_nowires.fcl -s /home/hamlet/physics/LArSoft/data/production/develop/profiling/input/prod_piplus_0.1-2.0GeV_isotropic_3window_uboone_15314131_91_gen_15314662_91_g4_15342662_91_detsim_tpc.root -n 3 (standard_reco_uboone_2D_noo…
```

Stacked diagrams: 10

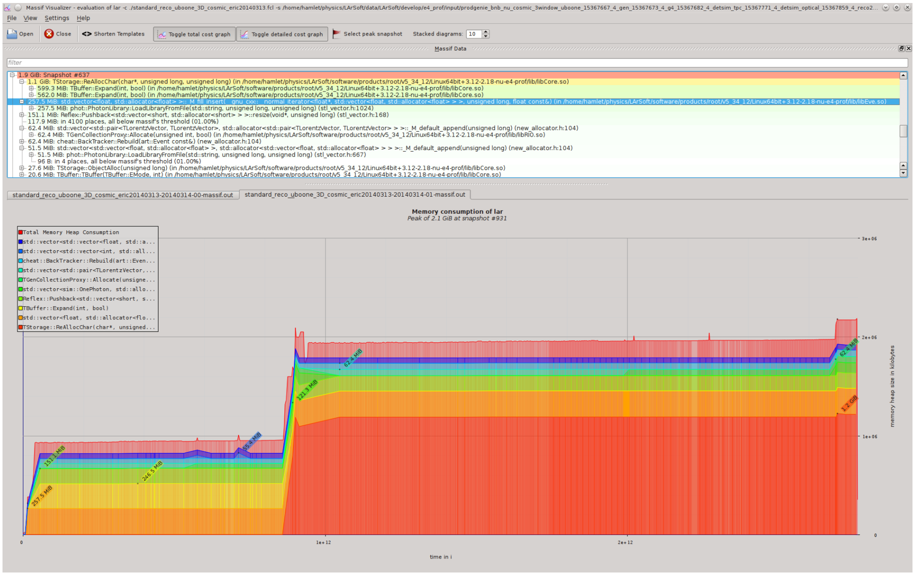
```
1.7 GiB: Snapshot #321 (peak)
  608.0 MiB: TStorage::ReAllocChar(char*, unsigned long, unsigned long) (in /home/hamlet/physics/LArSoft/software/products/root/v5_34_12/Linux64bit+3.13-2.17-nu-e4-prof/lib/libCore.so)
  450.0 MiB: recob::Wire* std::__uninitialized_copy<false>::__uninit_copy<recob::Wire*, recob::Wire*>(recob::Wire*, recob::Wire*, recob::Wire*) (new_allocator.h:104)
    450.0 MiB: void std::vector<recob::Wire, std::allocator<recob::Wire> >::_M_emplace_back_aux<recob::Wire>(recob::Wire&&) (stl_uninitialized.h:117)
      450.0 MiB: caldata::CalWireMicroBooNE::produce(art::Event&) (stl_vector.h:920)
  331.1 MiB: TArrayF::Set(int) (in /home/hamlet/physics/LArSoft/software/products/root/v5_34_12/Linux64bit+3.13-2.17-nu-e4-prof/lib/libCore.so)
    330.6 MiB: TH2F::TH2F(char const*, char const*, int, double, double, int, double, double) (in /home/hamlet/physics/LArSoft/software/products/root/v5_34_12/Linux64bit+3.13-2.17-nu-e4-prof/lib/libHist.so)
      302.6 MiB: cluster::CornerFinderAlg::TakeInRaw(art::Event const&) (CornerFinderAlg.cxx:213)
      28.0 MiB: in 4 places, all below massif's threshold (01.00%)
    586.0 KiB: in 3 places, all below massif's threshold (01.00%)
  151.1 MiB: Reflex::Pushback<std::vector<short, std::allocator<short> > >::resize(void*, unsigned long) (stl_vector.h:168)
  150.0 MiB: void std::vector<recob::Wire, std::allocator<recob::Wire> >::emplace_back<recob::Wire>(recob::Wire&&) (new_allocator.h:104)
    150.0 MiB: caldata::CalWireMicroBooNE::produce(art::Event&) (stl_vector.h:920)
  56.7 MiB: in 4071 places, all below massif's threshold (01.00%)
```

standard_reco_uboone_2D_noopt_nowires-20140311-00-massif.out

Memory consumption of lar
Peak of 1.7 GiB at snapshot #321

File
View
Settings
Help
Open
Close
Shorten Templates
Toggle total cost graph
Toggle detailed cost graph
Select peak snapshot
Massif Data
filter
Total Memory Heap Consumption
std::vector<sim::OnePhoton, std::allo...
TArrayD::Set(int)
TStorage::ObjectAlloc(unsigned long)
void std::vector<recob::Wire, std::al...
Reflex::Pushback<std::vector<short, s...
TBuffer::Expand(int, bool)
cluster::HoughTransform::AddPointRetu...
TArrayF::Set(int)
recob::Wire* std::__uninitialized_cop...
TStorage::ReAllocChar(char*, unsigned...
1.7 GiB
150.0 MiB
450.0 MiB
199.0 MiB
151.1 MiB
163.7 MiB
19.8 MiB
25.7 MiB
350.4 MiB
18.6 MiB
653.7 MiB
2e+06
1.8e+06
1.6e+06
1.4e+06
1.2e+06
1e+06
800000
600000
400000
200000
0
memory heap size in kilobytes
0
1e+11
2e+11
3e+11
4e+11
5e+11
6e+11

massif example: memory fragmentation? (reco 3D)



Massif Visualizer - evaluation of lar -c ./standard_reco_uboone_3D_cosmic_eric20140313.fcl -s /home/hamlet/physics/LArSoft/data/LArSoft/develop/e4_prof/input/prodgenie_bnb_nu_cosmic_3window_uboone_15367667_4_gen_15367673_4_g4_15367682_4_detsim_tpc_15367771_4_detsim_optical_15367859_4_reco2...

Stacked diagrams: 10

```
⊟ 1.9 GiB: Snapshot #637
  ⊟ 1.1 GiB: TStorage::ReAllocChar(char*, unsigned long, unsigned long) (in /home/hamlet/physics/LArSoft/software/products/root/v5_34_12/Linux64bit+3.12-2.18-nu-e4-prof/lib/libCore.so)
    ⊞ 599.3 MiB: TBuffer::Expand(int, bool) (in /home/hamlet/physics/LArSoft/software/products/root/v5_34_12/Linux64bit+3.12-2.18-nu-e4-prof/lib/libCore.so)
    ⊞ 562.0 MiB: TBuffer::Expand(int, bool) (in /home/hamlet/physics/LArSoft/software/products/root/v5_34_12/Linux64bit+3.12-2.18-nu-e4-prof/lib/libCore.so)
  ⊟ 257.5 MiB: std::vector<float, std::allocator<float> >::_M_fill_insert(__gnu_cxx::__normal_iterator<float*, std::vector<float, std::allocator<float> > >, unsigned long, float const&) (in /home/hamlet/physics/LArSoft/software/products/root/v5_34_12/Linux64bit+3.12-2.18-nu-e4-prof/lib/libEve.so)
    ⊞ 257.5 MiB: phot::PhotonLibrary::LoadLibraryFromFile(std::string, unsigned long, unsigned long) (stl_vector.h:1024)
  ⊞ 151.1 MiB: Reflex::Pushback<std::vector<short, std::allocator<short> > >::resize(void*, unsigned long) (stl_vector.h:168)
  - 117.9 MiB: in 4100 places, all below massif's threshold (01.00%)
  ⊟ 62.4 MiB: std::vector<std::pair<TLorentzVector, TLorentzVector>, std::allocator<std::pair<TLorentzVector, TLorentzVector> > >::_M_default_append(unsigned long) (new_allocator.h:104)
    ⊞ 62.4 MiB: TGenCollectionProxy::Allocate(unsigned int, bool) (in /home/hamlet/physics/LArSoft/software/products/root/v5_34_12/Linux64bit+3.12-2.18-nu-e4-prof/lib/libRIO.so)
  ⊞ 62.4 MiB: cheat::BackTracker::Rebuild(art::Event const&) (new_allocator.h:104)
  ⊟ 51.5 MiB: std::vector<std::vector<float, std::allocator<float> >, std::allocator<std::vector<float, std::allocator<float> > > >::_M_default_append(unsigned long) (new_allocator.h:104)
    ⊞ 51.5 MiB: phot::PhotonLibrary::LoadLibraryFromFile(std::string, unsigned long, unsigned long) (stl_vector.h:667)
    - 96 B: in 4 places, all below massif's threshold (01.00%)
  ⊞ 27.6 MiB: TStorage::ObjectAlloc(unsigned long) (in /home/hamlet/physics/LArSoft/software/products/root/v5_34_12/Linux64bit+3.12-2.18-nu-e4-prof/lib/libCore.so)
  ⊞ 20.6 MiB: TBuffer::TBuffer(TBuffer::EMode, int) (in /home/hamlet/physics/LArSoft/software/products/root/v5_34_12/Linux64bit+3.12-2.18-nu-e4-prof/lib/libCore.so)
```

standard_reco_uboone_3D_cosmic_eric20140313-20140314-00-massif.out | standard_reco_uboone_3D_cosmic_eric20140313-20140314-01-massif.out

Memory consumption of lar
Peak of 2.1 GiB at snapshot #931

File
View
Settings
Help
Open
Close
<> Shorten Templates
Toggle total cost graph
Toggle detailed cost graph
Select peak snapshot
Massif Data
filter
🟥 Total Memory Heap Consumption
🟦 std::vector<std::vector<float, std::a...
🟦 std::vector<std::vector<int, std::all...
🟦 cheat::BackTracker::Rebuild(art::Even...
🟩 std::vector<std::pair<TLorentzVector,...
🟩 TGenCollectionProxy::Allocate(unsigne...
🟩 std::vector<sim::OnePhoton, std::allo...
🟩 Reflex::Pushback<std::vector<short, s...
🟨 TBuffer::Expand(int, bool)
🟧 std::vector<float, std::allocator<flo...
🟥 TStorage::ReAllocChar(char*, unsigned...
151.1 MiB
257.5 MiB
246.5 MiB
55.4 MiB
62.4 MiB
121.3 MiB
62.4 MiB
1.2 GiB
3e+06
2e+06
1e+06
0
memory heap size in kilobytes
0
1e+12
2e+12
time in i

Profiling times

Currently, I can run with `e4:prof` code:

- plain run: 100-event samples (thousands should not be a problem)
- statistical CPU speed profiling: few percent overhead respect to plain
- complete call profiling: 5-event chunks (takes 40'/1h each; 10 possible)
- memory profiling: roughly as complete call profiling (it's `valgrind`)
- memory and stack profiling: 3-event chunks (takes longer than just memory)

A 8 GB memory machine would help making this quicker (virtual memory is deadly — for many people, if the machine is shared).

Stack profiling has shown to be not necessary (fortunately!).

Outlook

- each profiling tool has its strengths
- they must be understood, or wrong conclusions will be drawn
- best coding practises should be routinely applied (not "at the end")
- testing is an important part of both the development and the optimization process

Supplemental material

Best coding practises?

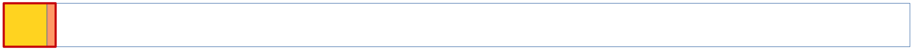
I am no coding authority, so my opinion follows:

- design and implement a test together with the code
- use dynamic allocation (`new` operator) only if a pointer to the data is needed
- never allow “naked” pointer to exit the creation scope
- when writing large-iteration loops, move the constant things out of them
- for the analysis of a lot of items, apply all the analysis steps to one item, rather than performing one step on all items
- beware of continuous reallocation of memory (`std::vector::push_back`), reserve the memory beforehand when possible
- reduce the copies of objects: use references, in-place construction (`std::vector::emplace_back`) or, if possible, `std::move`

What is memory fragmentation

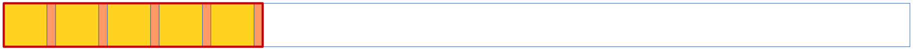
```
std::vector<TMatrixD> M;
std::vector<TVectorD> V;
for (size_t i = 0; i < BigNumber; ++i) {
   M.push_back(TMatrixD(36));
   V.push_back(TVectorD(36));
}
M.clear();
double* data = new double[BigBummer];
```

What is memory fragmentation



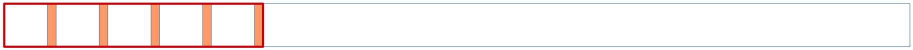
```
std::vector<TMatrixD> M;
std::vector<TVectorD> V;
for (size_t i = 0; i < BigNumber; ++i) {
   M.push_back(TMatrixD(36));
   V.push_back(TVectorD(36));
}
M.clear();
double* data = new double[BigBummer];
```

What is memory fragmentation



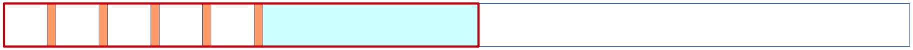
```
std::vector<TMatrixD> M;
std::vector<TVectorD> V;
for (size_t i = 0; i < BigNumber; ++i) {
   M.push_back(TMatrixD(36));
   V.push_back(TVectorD(36));
}
M.clear();
double* data = new double[BigBummer];
```

What is memory fragmentation



```
std::vector<TMatrixD> M;
std::vector<TVectorD> V;
for (size_t i = 0; i < BigNumber; ++i) {
   M.push_back(TMatrixD(36, 36));
   V.push_back(TVectorD(36));
}
M.clear();
double* data = new double[BigBummer];
```

What is memory fragmentation



```
std::vector<TMatrixD> M;
std::vector<TVectorD> V;
for (size_t i = 0; i < BigNumber; ++i) {
   M.push_back(TMatrixD(36));
   V.push_back(TVectorD(36));
}
M.clear();
double* data = new double[BigBummer];
```

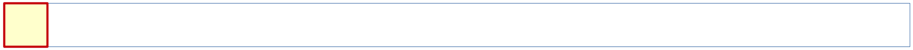
The red rectangle outlines the heam memory usage of the program.
Note the white holes of unused memory.

Why to use `std::vector::reserve`

```
std::vector<int> v;
v.push_back(1);
v.push_back(2);
v.push_back(3);
v.push_back(4);
```

We start with an empty vector.

Why to use `std::vector::reserve`

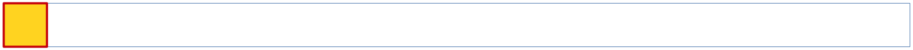


```
std::vector<int> v;
v.push_back(1);
v.push_back(2);
v.push_back(3);
v.push_back(4);
```

We push in the first element:

1. memory is allocated for the element

Why to use `std::vector::reserve`

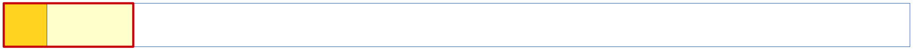


```
std::vector<int> v;
v.push_back(1);
v.push_back(2);
v.push_back(3);
v.push_back(4);
```

We push in the first element:

1. memory is allocated for the element
2. the object is *copied*

Why to use `std::vector::reserve`

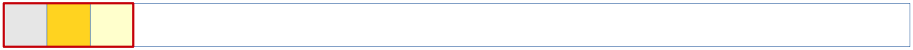


```
std::vector<int> v;
v.push_back(1);
v.push_back(2);
v.push_back(3);
v.push_back(4);
```

We push in the second element:

1. memory is allocated for two more elements (in fact, `gcc`'s STL vector doubles the space)

Why to use `std::vector::reserve`

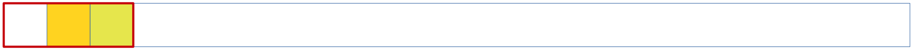


```
std::vector<int> v;
v.push_back(1);
v.push_back(2);
v.push_back(3);
v.push_back(4);
```

We push in the second element:

1. memory is allocated for two more elements (in fact, `gcc`'s STL vector doubles the space)
2. the existing object is *moved*

Why to use `std::vector::reserve`

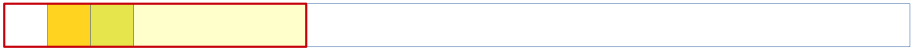


```
std::vector<int> v;
v.push_back(1);
v.push_back(2);
v.push_back(3);
v.push_back(4);
```

We push in the second element:

1. memory is allocated for two more elements (in fact, `gcc`'s STL vector doubles the space)
2. the existing object is *moved*
3. the old memory is freed; the new object is *copied*

Why to use `std::vector::reserve`

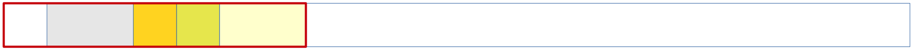


```
std::vector<int> v;
v.push_back(1);
v.push_back(2);
v.push_back(3);
v.push_back(4);
```

We push in the third element:

1. memory is allocated for four more elements (again doubling the space)

Why to use `std::vector::reserve`

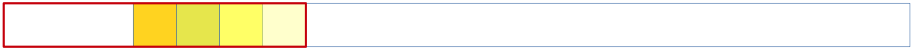


```
std::vector<int> v;
v.push_back(1);
v.push_back(2);
v.push_back(3);
v.push_back(4);
```

We push in the third element:

1. memory is allocated for four more elements (again doubling the space)
2. the existing objects are *moved*

Why to use `std::vector::reserve`

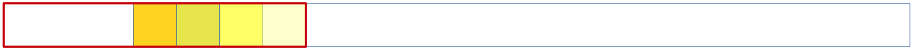


```
std::vector<int> v;
v.push_back(1);
v.push_back(2);
v.push_back(3);
v.push_back(4);
```

We push in the third element:

1. memory is allocated for four more elements (again doubling the space)
2. the existing objects are *moved*
3. the old memory is freed; the new object is *copied*

Why to use `std::vector::reserve`



```
std::vector<int> v;
v.push_back(1);
v.push_back(2);
v.push_back(3);
v.push_back(4);
```

We had:

- 3 copy-constructed objects
- 3 memory allocations
- 2 memory deallocations
- 2 “fast” copies (*possibly* not involving copy constructors)
- some fragmented memory in the end

Had we used a `v.reserve(4)` call after defining the vector:

- 3 copy-constructed objects
- 1 memory allocation
- compact memory in the end